Creating the Unity Container

```
class Program
{
  static void Main(string[] args)
  {
    using (var container = new UnityContainer())
    {
       //configure container

       //ask container for objects (configured) and use them
    }
```

Configuring the Unity Container

```
class Program
{
  static void Main(string[] args)
  {
    using (var container = new UnityContainer())
    {
       //configure container
       container
         .RegisterType<IOutputFormatter, OutputFormatter>()
         .RegisterType<IPrimeGenerator, PrimeGenerator>();

       //ask container for objects (configured) and use them
    }

  }
}
```

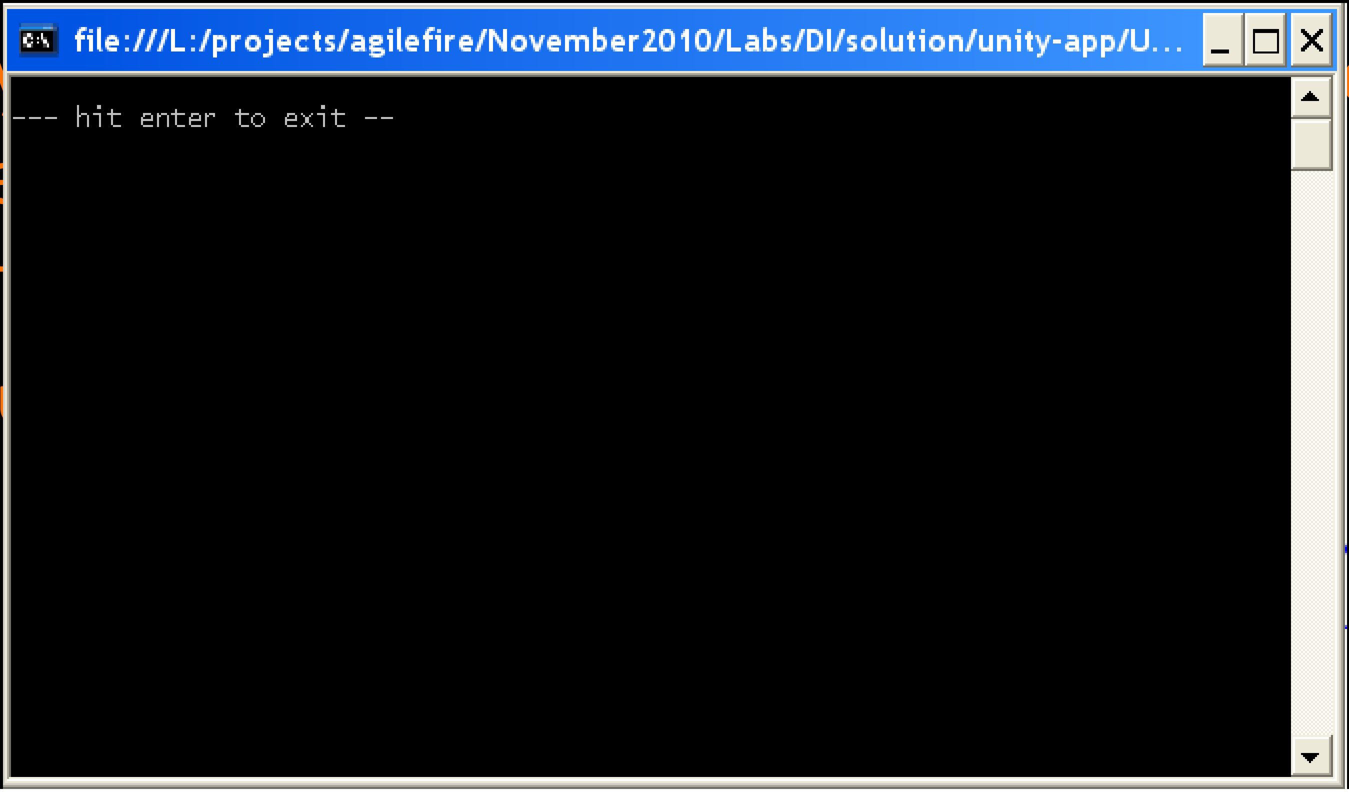
Getting Configured Objects

```
class Program {
  static void Main(string[] args) {
    using (var container = new UnityContainer()){

      //configure container
      container
        .RegisterType<IOutputFormatter, OutputFormatter>()
        .RegisterType<IPrimeGenerator, PrimeGenerator>();

      //ask container for objects (configured) and use them
      var report = container.Resolve<ConsoleReport>();
      report.Write();
    }
  }
}
```

Agile Firestarter

However…

- Runs but no output ☹
  - Default max number is set to 0
- Onl ect type
  - ‘T
  - B



file:///L:/projects/agilefire/November2010/Labs/DI/solution/unity-app/U...

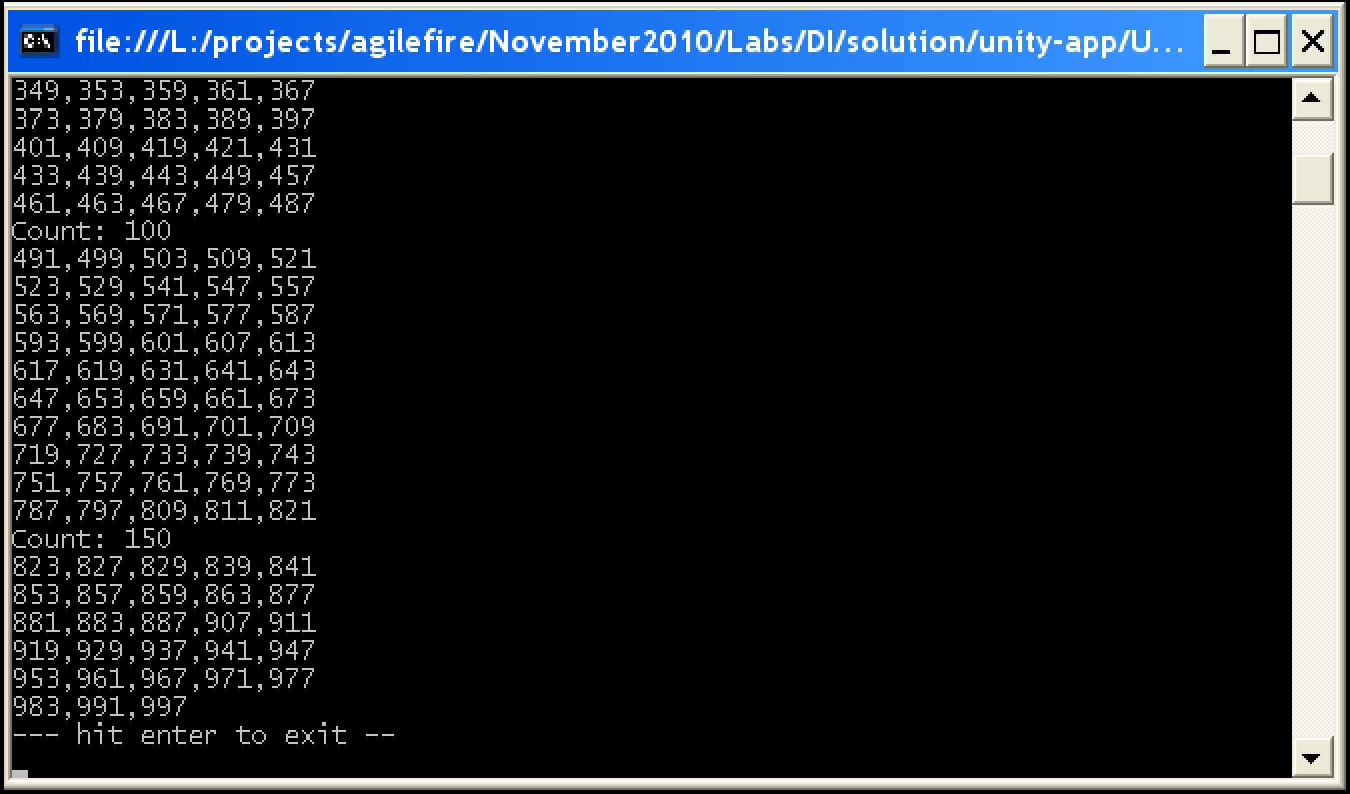
```
--- hit enter to exit --
```

Configuring Simple Properties

```
class Program {
  static void Main(string[] args) {
    using (var container = new UnityContainer()) {

      //configure container
      container
        .RegisterType<IOutputFormatter, OutputFormatter>()
        .RegisterType<IPrimeGenerator, PrimeGenerator>();

      container.Configure<InjectedMembers>()
               .ConfigureInjectionFor<ConsoleReport>(
                    new InjectionProperty("MaxNumber", 1000)
                );
      //ask container for objects (configured) and use them
      var report = container.Resolve<ConsoleReport>();
      report.Write();
    }
  }
}
```

Agile Firestarter

Now we get our prime numbers…


file:///L:/projects/agilefire/November2010/Labs/DI/solution/unity-app/U...

```
349,353,359,361,367
373,379,383,389,397
401,409,419,421,431
433,439,443,449,457
461,463,467,479,487
Count: 100
491,499,503,509,521
523,529,541,547,557
563,569,571,577,587
593,599,601,607,613
617,619,631,641,643
647,653,659,661,673
677,683,691,701,709
719,727,733,739,743
751,757,761,769,773
787,797,809,811,821
Count: 150
823,827,829,839,841
853,857,859,863,877
881,883,887,907,911
919,929,937,941,947
953,961,967,971,977
983,991,997
--- hit enter to exit --
```

Object Lifecycle

- What happens if you call Resolve twice…?
- Container has options to control the lifecycle
  - Transient/“Prototype” – new one each time
    - The default option
  - Singleton – one for the life of the container
    - `ContainerControlledLifetimeManager`
  - Externally Controlled – one for the life of the container
    - But container holds a weak reference to the object
    - `ExternallyControlledLifetimeManager`
  - `LifetimeManager` base class to customize

Now as singletons…

```
class Program {
  static void Main(string[] args) {
    using (var container = new UnityContainer())
    {
       //configure container
        container
         .RegisterType<IOutputFormatter, OutputFormatter>(new
                                   ContainerControlledLifetimeManager())
         .RegisterType<IPrimeGenerator, PrimeGenerator>(new
                                   ContainerControlledLifetimeManager())
         .RegisterType<ConsoleReport>(new ContainerControlledLifetimeManager());

        container.Configure<InjectedMembers>()
                .ConfigureInjectionFor<ConsoleReport>(
                    new InjectionProperty("MaxNumber", 1000)
                );
       //ask container for objects (configured) and use them
       var report = container.Resolve<ConsoleReport>();
       report.Write();
    }
  }
}
```

Setter Dependency Injection

```
class Program {
  static void Main(string[] args) {
    using (var container = new UnityContainer()) {
       //configure container
      container
        .RegisterType<IOutputFormatter, OutputFormatter>()
        .RegisterType<IEmailService,    EmailService>()
        .RegisterType<IPrimeGenerator,  PrimeGenerator>();

      container.Configure<InjectedMembers>()
                 .ConfigureInjectionFor<ConsoleReport>(
                      new InjectionProperty("EmailService",
                          new ResolvedParameter<IEmailService>())
                 );
       //ask container for objects (configured) and use them
       var report = container.Resolve<ConsoleReport>();
       report.Write();
     }
  }
}
```

Setter Dependency Injection (II)

```
public class ConsoleReport
{
  private IEmailService _emailService;
  private IOutputFormatter _outputFormatter;
  private IPrimeGenerator _primeGenerator;

  public ConsoleReport(IOutputFormatter outputFormatter,
                       IPrimeGenerator primeGenerator)
  {
    _outputFormatter = outputFormatter;
    _primeGenerator = primeGenerator;
  }

  [Dependency]
  public IEmailService EmailService
  {
    set { _emailService = value; }
  }
}
```

No longer a POCO